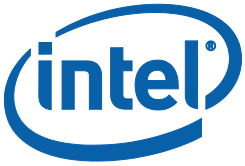


# Intel® OpenCL SDK

## User's Guide

Copyright © 2010–2011 Intel Corporation

All Rights Reserved

Document Number: 323626-003US

Revision: 1.3

World Wide Web: http://www.intel.com

## Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.
Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to http://www.intel.com/design/literature.htm.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel, Intel logo, Intel Core, VTune, Xeon are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright © 2010-2011 Intel Corporation. All rights reserved.

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# Contents


1 Introduction ..... 6

1.1 OpenCL™ Overview ..... 6

1.2 About Intel® OpenCL SDK ..... 6

1.3 Conventions and Symbols ..... 7

1.4 Related Information ..... 7

2 Intel® OpenCL SDK Features ..... 8

2.1 OpenCL 1.1 Conformant Extensions ..... 8

2.2 OpenCL cl_ext Extensions ..... 9

2.2.1 Intel Device Fission Extension Support ..... 9

2.2.2 Intel Immediate Command Execution Extension Support ..... 10

2.3 Intel Vendor Extensions ..... 11

2.3.1 Intel printf Extension ..... 11

2.4 OpenCL Installable Client Driver (ICD) ..... 11

2.5 Smart Compilation Using the Implicit CPU Vectorization Module ..... 12

2.6 Efficient, Highly Scalable Threading System ..... 12

2.7 Intel® VTune™ Amplifier XE 2011 Integration ..... 12

2.8 Intel® OpenCL SDK Offline Compiler ..... 12

2.9 Intel® Graphics Performance Analyzers ..... 13

2.10 Intel® OpenCL SDK Debugger ..... 13

3 Intel® OpenCL SDK Package Contents ..... 14

3.1 Package Contents for Microsoft* Windows* Operating Systems ..... 14

3.2 Package Contents for Linux* Operating Systems ..... 15

4 Using Intel® OpenCL SDK ..... 16

4.1 Using the OpenCL Runtime on Microsoft* Windows* Operating Systems ..... 16

4.1.1 Configuring Microsoft* Visual Studio* 2008 ..... 16

4.1.2 Using OpenCL Project Property Pages ..... 17

4.2 Using the OpenCL Runtime on Linux* Operating System ..... 18

4.3 Using Intel® OpenCL SDK Samples ..... 18

4.3.1 Building a Sample Application ..... 18

4.3.2 Running a Sample Application ..... 19

4.4 Working with the OpenCL Installable Client Driver (ICD) ..... 20
4.5 Working with the *cl-fast-relaxed-math* Flag ..... 21

5 Debug and Analyze with the Intel® OpenCL SDK Tools ..... 22

5.1 Analyzing Your Application with the Intel® VTune™ Amplifier XE 2011 ..... 22
5.1.1 Setting a New Profiling Project ..... 22
5.1.2 Viewing the OpenCL Kernel's Assembly Code ..... 23
5.2 Compiling Your Program with the Intel® OpenCL SDK Offline Compiler ..... 24
5.2.1 Building the OpenCL Kernels ..... 25
5.2.2 Viewing the Generated Assembly Code ..... 27
5.2.3 Choosing a Different Target Instruction Set Architecture ..... 28
5.2.4 Viewing the Generated LLVM Code ..... 28
5.2.5 Generating Intermediate Program Binaries ..... 29
5.2.6 Saving Your Code to a Text File ..... 29
5.2.7 Working with the Command Prompt ..... 30
5.3 Tracing OpenCL Commands with the Intel® Graphics Performance Analyzers 30
5.3.1 Generating a Trace File ..... 31
5.3.2 OpenCL Device View ..... 33
5.3.3 Controlling the Markers Display ..... 34
5.3.4 OpenCL Context View ..... 34
5.3.5 OpenCL API Tracing ..... 35
5.4 Using the Intel® OpenCL SDK Debugger ..... 36
5.4.1 Debugging Your OpenCL Kernel with Intel® OpenCL SDK Debugger 36
5.4.2 Configuring Intel® OpenCL SDK Debugger ..... 37
5.4.3 Troubleshooting the Intel® OpenCL SDK Kernel Debugger ..... 38

# 1 Introduction

The Intel® OpenCL SDK User's Guide contains the general information about OpenCL, Intel® OpenCL SDK 1.5, its features, tools, package contents, and basic usage guidelines.

## 1.1 OpenCL™ Overview

OpenCL™ (Open Computing Language) is the first open, royalty-free standard for general-purpose parallel programming of heterogeneous systems. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for client computer systems, high-performance computing servers, and handheld devices using a diverse mix of multi-core CPUs and other parallel processors.

## 1.2 About Intel® OpenCL SDK

Intel® OpenCL SDK is an implementation of the OpenCL standard optimized for Intel processors, running on Microsoft* Windows* and Linux* operating systems. Intel® OpenCL SDK version 1.5 supports the complete OpenCL 1.1 language and Application Programming Interface (API). Intel® OpenCL SDK 1.5 was validated with the Khronos* OpenCL conformance tests suite and is fully conformant with OpenCL 1.1 specification for the CPU and Microsoft* Windows* 7 operating systems.

For information on Intel® OpenCL SDK 1.5 limitations and known issues, please see the Intel® OpenCL SDK Release Notes [see Related Information].

# 1.3 Conventions and Symbols

The following conventions are used in this document.

**Conventions and Symbols used in this Document**

| | |
|---|---|
| `This type style` | Indicates an element of syntax, reserved word, keyword, filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant. |
| **This type style** | Indicates the exact characters you type as input. Also used to highlight the elements of a graphical user interface such as buttons and menu names. |

# 1.4 Related Information

By default Intel® OpenCL SDK Installation Notes, Release Notes and Writing Optimal OpenCL Code with Intel® OpenCL SDK documents are in:

> `C:\Program Files\Intel\OpenCL SDK\1.5\doc` (on 64-bit operating systems, instead of "`Program Files`", the directory name is "`Program Files (x86)`")

Intel® OpenCL SDK related information is also available at the Intel® OpenCL SDK official website.

For any information about OpenCL and Intel® OpenCL SDK, please refer to the Intel® OpenCL SDK Online Resources table:

| Description | Link |
|---|---|
| The OpenCL official page on the Khronos* website. All information about latest versions of OpenCL. | http://www.khronos.org/opencl/ |
| Information on the Khronos* conformance test process | http://www.khronos.org/adopters/ |
| The Intel® OpenCL SDK product page | http://software.intel.com/en-us/articles/intel-opencl-sdk/ |
| The Intel® OpenCL SDK download page | http://software.intel.com/en-us/articles/download-intel-opencl-sdk/ |
| The Intel® OpenCL SDK discussion and support forum | http://software.intel.com/en-us/forums/intel-opencl-sdk/ |

# 2 Intel® OpenCL SDK Features

Intel® OpenCL SDK 1.5 supports the following OpenCL 1.1 optional features, as defined in the OpenCL specification:

- Out-of-order execution model (`CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` property of a command-queue).
- Execution of native kernels (`CL_EXEC_NATIVE_KERNEL` option of the `CL_DEVICE_EXECUTION_CAPABILITIES` property of device information).
- Image support with the basic set of image formats (`CL_DEVICE_IMAGE_SUPPORT` property of device information).
- Optimization Options: SDK supports the OpenCL 1.1 standard compiler flag `-cl-fast-relaxed-math`. Some optimizations may violate the IEEE 754 standard and the OpenCL numerical compliance. For a full list of optimized functions see Working with the cl-fast-relaxed-math Flag.
- Math intrinsic option: SDK supports the OpenCL 1.1 standard optional compiler flag `-cl-denorms-are-zero`

## 2.1 OpenCL 1.1 Conformant Extensions

Intel® OpenCL SDK 1.5 includes fully conformant implementation of the following OpenCL 1.1 extensions, as defined in the OpenCL specification:

- `cl_khr_fp64` - double precision floating point support.
- `cl_khr_gl_sharing` - creating OpenCL context from an OpenGL* context or share group on Microsoft* Windows* OS only.
- `cl_khr_gl_sharing` - sharing memory objects with OpenGL* or OpenGL* ES buffers, texture and render bugger objects on Microsoft* Windows* OS only.

# 2.2 OpenCL cl_ext Extensions

## 2.2.1 Intel Device Fission Extension Support

Intel® OpenCL SDK 1.5 supports the `cl_ext_device_fission` extension. Device Fission is an extension which enables you to control compute unit utilization within a compute device using the SDK. Intel® OpenCL SDK 1.5 supports the following fission modes:

- `CL_DEVICE_PARTITION_EQUALLY_EXT` enables you to create equally-sized sub-devices, each containing the provided number of compute units. Remainder compute units of the parent device are not used.
- `CL_DEVICE_PARTITION_BY_COUNTS_EXT` enables you to specify a list of sizes. A sub-device of the appropriate size is created for every item on the list. The sum of sizes provided for sub-devices must not be more than the size available to the parent device.
- `CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN_EXT` enables you to split a device according to an architectural feature of the parent device. Intel® OpenCL SDK 1.5 supports only the `CL_AFFINITY_DOMAIN_NUMA_EXT` property, which creates a sub-device for every Non-Uniform Memory Access (NUMA) node on the platform.

Device Fission enables improving the performance of the applications which use OpenCL. To gain the most from this feature, follow these guidelines:

- There is an overhead which depends on the amount of created sub devices. Use `clCreateSubdevicesEXT` to create exactly as many sub-devices as you are going to use in your program. To create a sub-device with half the compute units of a parent, use `BY_COUNTS` with a single item to create only one sub-device instead of using `EQUALLY` that creates two. The fewer sub-devices you create, the more performance you gain.
- Avoid simultaneously executing commands in the following configurations
    - on a sub-device and its ancestor
    - on sub-devices that could overlap compute units

- mixed sub-devices from different partitioning modes
- mixing sub-devices from different calls to `clCreateSubDevicesEXT` that must overlap if executed simultaneously. For example, sub-devices created `BY_COUNTS` totaling more than the amount of available compute units on the parent device.

- According to the OpenCL language semantics, you can not inform the runtime on which NUMA node memory object should be created. Using `CL_AFFINITY_DOMAIN_NUMA_EXT` property, you achieve best performance, ensuring allocation of the physical pages on the correct node, and creating memory objects with the `USE_HOST_PTR` property

### 2.2.2 Intel Immediate Command Execution Extension Support

Intel® OpenCL SDK 1.5 supports the `cl_intel_immediate_execution` extension. Immediate Command Execution is an extension which enables you to execute OpenCL commands in a single-threaded manner, using the calling thread to perform the actual execution.

To utilize this extension, add the token `CL_QUEUE_IMMEDIATE_EXECUTION_ENABLE_INTEL` to the queue properties at `clCreateCommandQueue` time. `clEnqueueXXX` calls to that queue are synchronous – they return only after the enqueued command finishes executing. Moreover, only the thread calling the `clEnqueueXXX` call executes those commands. This includes calls to `clEnqueueNDRange`.

Using this extension, you can create a command queue alongside the rest of the queues, and use it to execute lightweight kernels or NDRanges with a high granularity (small global size), that cannot gain much from the benefits of Intel multi-core architecture. You will still get the full benefits of the Intel® OpenCL SDK OpenCL compiler, including the automatic vectorization module.

Please note an Immediate Command Execution queue can still be in-order or out-of-order. In the in-order mode, if multiple threads enqueue to the same queue at the same time, they block each other to comply with the OpenCL in-order queue semantics. So, you should prefer using the combination of `CL_QUEUE_IMMEDIATE_EXECUTION_ENABLE_INTEL` and `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE`.

# 2.3 Intel Vendor Extensions

## 2.3.1 Intel printf Extension

Intel® OpenCL SDK 1.5 provides the `cl_intel_printf` extension, which enables you to use the C `printf` function with support of OpenCL vector types. The `printf` function can be seen as a new OpenCL built-in function with the following signature:

```
int printf(__constant char* restrict format, …).
```

The semantics of `printf` match the definitions found in section 7.19.6 of the C99 standard, with the following notes:

- 64-bit integer types can be printed using the `l` (lowercase L) length modifier (e.g. `%lu`)
- The `ll` length modifier is not supported.

The OpenCL vector types can be explicitly passed and printed using the modifier vn where n can be `2, 3, 4, 8` or `16`.

***NOTE:*** The vn modifier appears before the original conversion specifier. To print a `int4` the conversion specifier is `%v4d`. Use commas to separate printed vector components.

# 2.4 OpenCL Installable Client Driver (ICD)

The OpenCL Installable Client Driver (ICD) enables multiple OpenCL implementations to coexist under the same system. ICD also enables applications to select between OpenCL implementations at run time.

Intel® OpenCL SDK supports the OpenCL 1.1 ICD. See Working with the OpenCL Installable Client Driver (ICD) for more details.

## 2.5 Smart Compilation Using the Implicit CPU Vectorization Module

Intel® OpenCL SDK provides a smart OpenCL compilation environment which efficiently maps the code to processor vector units and includes a unique implicit CPU vectorization module for best utilization of the hardware vector units across work items. The vectorization module aims to merge together the execution of several work items, utilizing the Intel vector instruction set.

The compilation environment is based on the open source LLVM compiler and its clang front end.

## 2.6 Efficient, Highly Scalable Threading System

Intel® OpenCL SDK contains an efficient, highly scalable threading system for optimal multi-core performance. The system is based on the Intel® Threading Building Blocks (Intel® TBB).

## 2.7 Intel® VTune™ Amplifier XE 2011 Integration

Intel® OpenCL SDK 1.5 enables you to analyze the assembly code of the OpenCL kernel on Microsoft* Windows* operating systems only with the Intel® VTune™ Amplifier XE 2011. For more information, see Analyzing Your Application with the Intel® VTune™ Amplifier XE 2011.

## 2.8 Intel® OpenCL SDK Offline Compiler

Intel® OpenCL SDK provides a unique standalone tool which offers full offline OpenCL language compilation including the OpenCL syntax checker, LLVM (Low Level Virtual Machine) viewer, Assembly language viewer, cross hardware platform compilation and intermediate program binaries generator. See Compiling Your Application with the Intel® OpenCL SDK Offline Compiler for more details.

## 2.9 Intel® Graphics Performance Analyzers

Intel® OpenCL SDK provides a built-in instrumentation of the Intel® GPA Platform Analyzer tool which enables you to see the OpenCL kernel execution data in a trace. You can analyze the trace offline to enhance the performance of your OpenCL application. See Tracing OpenCL Commands with Intel® Graphics Performance Analyzers for more details. Intel® Graphics Performance Analyzers (Intel® GPA) support is available on Microsoft* Windows* 7 and Microsoft* Windows Vista* operating systems only.

## 2.10 Intel® OpenCL SDK Debugger

Intel® OpenCL SDK 1.5 provides a plug-in for Microsoft* Visual Studio* 2008 which enables you to debug your OpenCL kernels. For more information, see Using the Intel® OpenCL SDK Debugger.

# 3 Intel® OpenCL SDK Package Contents

Intel® OpenCL SDK runs on Microsoft* Windows* and Linux* operating systems, and consequently has different installation packages for both families of operating systems. The next two sections describe package contents for Windows* and Linux* operating systems.

## 3.1 Package Contents for Microsoft* Windows* Operating Systems

Intel® OpenCL SDK package for Windows* operating systems contains SDK libraries, documentation, developer files, tools and samples. During the installation on a Microsoft* Windows* OS system, you can choose the installation path for the libraries and samples.

The default installation path on Microsoft* Windows* operating systems is:

- `C:\Program Files\Intel\OpenCL SDK` (on 64-bit operating systems, instead of "`Program Files`", the directory name is "`Program Files (x86)`"

The default installation path for the samples is:

- `C:\Users\<user name>\Documents\Intel\OpenCL SDK`

Where <user name> is the name of your local user account, or "Public" if the product is installed for all user accounts.

Installation directory has the following structure:

- `bin` directory contains the OpenCL-related binaries
- `lib` directory contains the basic OpenCL run-time library and the OpenCL ICD library, to which the applications should link.
- `include` directory contains relevant header files, including the headers for the OpenCL runtime
- doc directory includes the Intel® OpenCL SDK documentation:
  - *Intel(R) OpenCL SDK Installation Notes.pdf*

  - *Intel(R) OpenCL SDK User Guide.pdf*
  - *Intel(R) OpenCL SDK Release Notes.pdf*
  - *Writing Optimal OpenCL(tm) Code with Intel(R) OpenCL SDK.pdf*
- `tools` directory includes the tools which come with SDK. The tool-set includes Intel® OpenCL SDK Offline Compiler.
- `samples` directory includes sample projects and property pages. The samples package includes the following samples (including sample documentation):
  - *Dot Product* demonstrates how to compute a dot product of two float4 arrays and writes the result to a float array.
  - *Bitonic Sort* demonstrates how to sort an arbitrary input array of integer values with OpenCL using Single Instruction Multiple Data (SIMD) bitonic sorting networks.
  - *God Rays* demonstrates how to use high dynamic range (HDR) rendering with God Rays (crepuscular rays) in OpenCL.
  - *Median Filter* demonstrates how to use a median filter in OpenCL.
  - *Shallow Water* demonstrates shallow water modeling to visualize a water surface.
  - *Tone Mapping* demonstrates how to use high dynamic range tone mapping post processing effect in OpenCL.
  - *Simple Optimizations* demonstrates several optimization methods including: using host pointers, relaxed math and auto work group size choice.
  - *Intel® Media SDK Interoperability* demonstrates how to use Intel® Media SDK and Intel® OpenCL SDK together for video processing.

## 3.2 Package Contents for Linux* Operating Systems

Intel® OpenCL SDK package for Linux* operating systems contains SDK libraries, developer files, and tools. Intel® OpenCL SDK for Linux* operating systems installer copies SDK files to the following folders:

- `/usr/include/CL/` directory contains relevant header files, including the headers for the OpenCL run time.
- `/usr/lib64/` directory contains the ICD shared library
- `/etc/OpenCL/vendors` directory contains the ICD registration file
- `/usr/lib64/OpenCL/vendors/intel/` directory contains the OpenCL-related binaries.

# 4 Using Intel® OpenCL SDK

## 4.1 Using the OpenCL Runtime on Microsoft* Windows* Operating Systems

Intel® OpenCL SDK 1.5 related binaries are installed to the following directory:

- `$(INTELOCLSDKROOT)\bin\x86` for 32-bit operation systems
- `$(INTELOCLSDKROOT)\bin\x64` for 64-bit operation systems

where the environment variable `INTELOCLSDKROOT` is automatically added to the system during installation and points to SDK installation directory.

This directory is automatically added to the system PATH environment. For more information, please see the Intel® OpenCL SDK Installation Notes [see Related Information].

To work with the OpenCL runtime, an application should link to the OpenCL Installable Client Driver (ICD) import library. The library is installed to:

- `$(INTELOCLSDKROOT)\lib\x86` for 32-bit operation systems
- `$(INTELOCLSDKROOT)\lib\x64` for 64-bit operation systems.

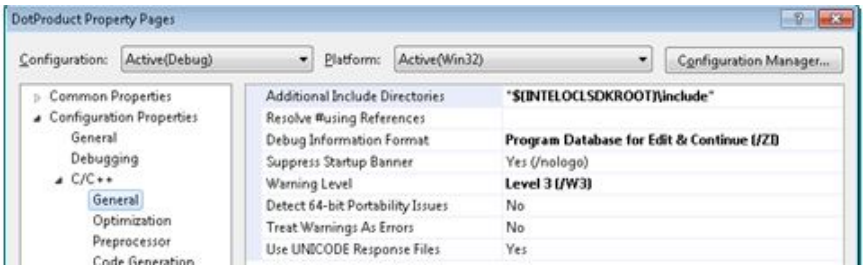
For more information on how to use OpenCL ICD, see Working with the OpenCL Installable Client Driver (ICD).

### 4.1.1 Configuring Microsoft* Visual Studio* 2008

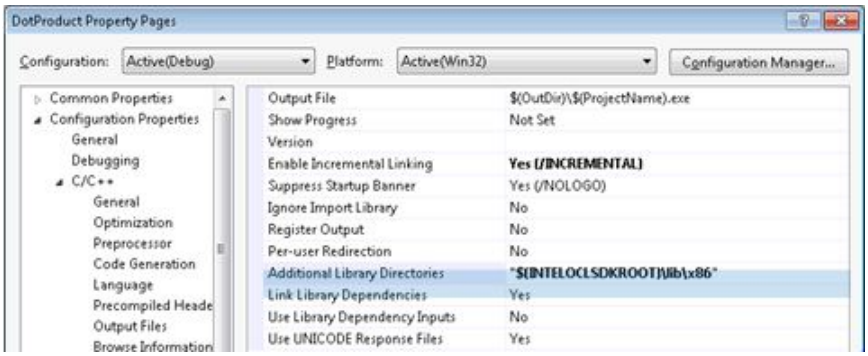
To setup a Microsoft* Visual Studio* 2008 project to work with Intel® OpenCL SDK 1.5:

1. Open the project property pages by selecting **Project** > **Properties**.
2. In the **C/C++** > **General** property page, under **Additional Include Directories**, enter the full path to the directory where the OpenCL header files are located:

`$(INTELOCLSDKROOT)\include`.




3. In the **Linker** > **General** property page, under **Additional Library Directories**, enter the full path to the directory where the OpenCL run-time import library file is located. For example, for 32-bit application: `$(INTELOCLSDKROOT)\lib\x86`




In the **Linker** > **Input** property page, under **Additional Dependencies**, enter the name of the OpenCL ICD import library file `OpenCL.lib`.

## 4.1.2 Using OpenCL Project Property Pages

A project property page is a Microsoft* Visual Studio* user interface element which enables you to specify and apply project settings to your projects.

Intel® OpenCL SDK 1.5 provides project property page templates with the required settings so you can easily create OpenCL applications. The templates were generated for the Microsoft* Visual Studio* 2008 software.

To set your project properties:

1. Create a new Microsoft* Visual Studio* project.
2. Open your project's **Property Pages** dialog box.

3. Using the **Configuration** dialog box, select configuration for which you want to set project properties, for example:
   - `intelocl_win32_debug.vsprops` for 32-bit debug configuration
   - `intelocl_win32_release.vsprops` for 32-bit release configuration.
4. Click **OK** to save the settings. If you want a change to take effect immediately, click **Apply**.

The property page template files are installed to the following directory:

`$(INTELOCLSAMPLESROOT)\templates`.

## 4.2 Using the OpenCL Runtime on Linux* Operating Systems

Intel® OpenCL SDK 1.5 related binaries are installed to the following directory:

`/usr/lib64/OpenCL/vendors/intel`.

To work with the OpenCL runtime, an application should link the application to the OpenCL Installable Client Driver (ICD), *`libOpenCL.so`*, which is installed to `/usr/lib64`.

For more information on how to use OpenCL ICD, see Working with the OpenCL Installable Client Driver (ICD).

## 4.3 Using Intel® OpenCL SDK Samples

Intel® OpenCL SDK Samples are supported only on Microsoft* Windows* operating systems.

### 4.3.1 Building a Sample Application

The Intel® OpenCL SDK 1.5 installation package contains a Microsoft* Visual Studio* 2008 solution file, `OpenCLSamples.sln`, residing in:

`$(INTELOCLSAMPLESROOT)\`.

INTELOCLSAMPLESROOT is an environment variable which is automatically added to the system during the installation and points to the Intel® OpenCL SDK samples package installation directory.

The solution file contains the following Microsoft* Visual Studio* 2008 sample projects: DotProduct, BitonicSort, GodRays, MedianFilter, ToneMapping, SimpleOptimizations.

The ShallowWater sample is available in a different solution ShallowWater.sln, residing in:

$(INTELOCLSAMPLESROOT)\ShallowWater\.

The Intel® Media SDK Interoperability sample is available in a different solution MediaSDKInterop.sln, residing in:

$(INTELOCLSAMPLESROOT)\MediaSDKInterop\.

To build all samples in the solution, open the solution files and select **Build** > **Build Solution**.

To build a specific project in the solution file, open OpenCLSamples.sln, right click on the project file in the Solution Explorer tree view and select **Build**.

## 4.3.2 Running a Sample Application

You can run a sample application using Microsoft* Visual Studio* 2008, or using the command line.

### 4.3.2.1 Using Microsoft* Visual Studio*

1. Open and build OpenCLSamples.sln**.**
2. Select a project file in the **Solution Explorer**.
3. Right-click the project and select **Set as StartUp Project**.

Press **Ctrl+F5** to run the application. To run the application in debug mode, press **F5**.

### 4.3.2.2 Using Command Line

1. Open a Microsoft* Visual Studio* command prompt.
2. Change to the directory according to the platform configuration on which you want to run the executable:
   - $(INTELOCLSAMPLESROOT)\Win32 for Win32 configuration
   - $(INTELOCLSAMPLESROOT)\x64 for x64 configuration.

3. Open the appropriate project configuration (**Debug** or **Release**).

Run the sample by entering the name of the executable. For example run `DotProduct.exe` to run the Dot Product sample.

## 4.4 Working with the OpenCL Installable Client Driver (ICD)

When using the OpenCL Installable Client Driver (ICD), the application is responsible for selecting the OpenCL platform to use. If there are several OpenCL platforms installed on the system, the application should use the `clGetPlatformIDs` and `clGetPlatformInfo` functions to query the available OpenCL platforms and decide which one to use.

The following example shows how to query the system platforms to get the correct platform ID:

```
   cl_platform_id * platforms = NULL;
   char vendor_name[128] = {0};
   cl_uint num_platforms = 0;

   // get number of available platforms
   cl_int err = clGetPlatformIDs(0, NULL, & num_platforms);
   if (CL_SUCCESS != err)
   {
      // handle error
   }
   platforms = (cl_platform_id*)malloc(
                              sizeof(cl_platform)*
num_platforms);
   if (NULL == platforms)
   {
      // handle error
   }
   err = clGetPlatformIDs(num_platforms, platforms, NULL);
   if (CL_SUCCESS != err)
   {
      // handle error
   }

   for (cl_uint ui=0; ui< num_platforms; ++ui)
   {
      err = clGetPlatformInfo(platforms[ui],
                              CL_PLATFORM_VENDOR,
                              128 * sizeof(char),
                              vendor_name,
```

```
                                    NULL);
      if (CL_SUCCESS != err)
      {
         // handle error
      }
      if (vendor_name != NULL)
      {
         if (!strcmp(vendor_name, "Intel Corporation"))
         {
            return platforms[ui];
         }
      }
   }
   // handle error
```

## 4.5 Working with the *cl-fast-relaxed-math* Flag

The `cl-fast-relaxed-math` flag sets the optimization options `-cl-finite-math-only` and `-cl-unsafe-math-optimizations` which enable optimizations for floating-point arithmetic that may violate the IEEE 754 standard and the OpenCL numerical compliance requirements defined in the OpenCL specification. This option defines the preprocessor macro `__FAST_RELAXED_MATH__` in the OpenCL program.

Intel® OpenCL SDK 1.5 provides the following optimizations to the `floating-point` version of the following built-in math functions:

- `gentype logb (gentype x)`
- `intn ilogb (gentype x)`
- `gentype fdim (gentype x, gentype y)`
- `gentype fmod (gentype x, gentype y)`
- `gentype hypot (gentype x, gentype y)`
- `gentype fract (gentype x, __global/__local/__private gentype *iptr)`
- `gentype sqrt (gentype)`
- `gentype rsqrt (gentype)`
- `gentype fmax (gentype x, gentype y)`
- `gentype fmin (gentype x, gentype y).`

# 5 Debug and Analyze with the Intel® OpenCL SDK Tools

## 5.1 Analyzing Your Application with the Intel® VTune™ Amplifier XE 2011

Intel® OpenCL SDK enables you to see the assembly code of your OpenCL kernels Just-in-time (JIT) code and to analyze its performance through sampling profiling (call-graph profiling is not supported in version 1.5) using the graphical interface of the Intel® VTune™ Amplifier XE performance profiler.

The Intel® VTune™ Amplifier XE 2011 works on Microsoft* Windows* and Linux* operating systems. You must acquire it separately. For more information, see the VTune Amplifier XE 2011 page at http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/.

***NOTE:*** The profiling support of Intel® OpenCL SDK is designed to work with the Intel® VTune™ Amplifier XE 2011 and Intel® VTune™ Performance Analyzer 9.1 or higher. Using other versions may cause undefined results.

### 5.1.1 Setting a New Profiling Project

The instructions below and all screen shots refer to the Intel® VTune™ Amplifier XE 2011. Take into consideration that user interface of the Intel® VTune™ Performance Analyzer may differ slightly.

To run profiling on the Intel® VTune™ Amplifier XE follow these steps:

1. Create a new sampling project by selecting **File** > **New** > **Project...**
2. In the **Create a Project** dialog, enter your new project's name and click **Create Project.**
3. In the **Project Properties** window, select the application to run, including the working directory and command arguments.

4. Click **Modify...** next to the user-defined environment variables text box and add two lines to the **User-defined Environment Variables** table:

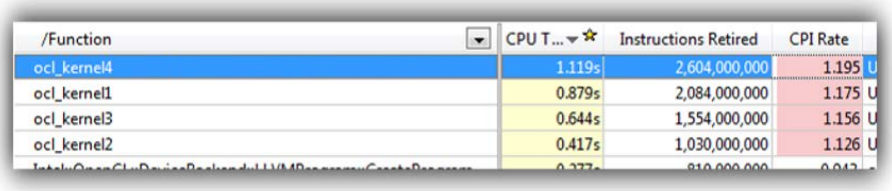
```
ENABLE_JITPROFILING=1
CL_CONFIG_USE_VTUNE=True
```

5. Click **OK** in all open windows to save the new settings.
6. Click the **New Analysis** button to run and analyze your application.
7. In the **Analysis Type** window, select the type you need and click **Start** to run the analysis.

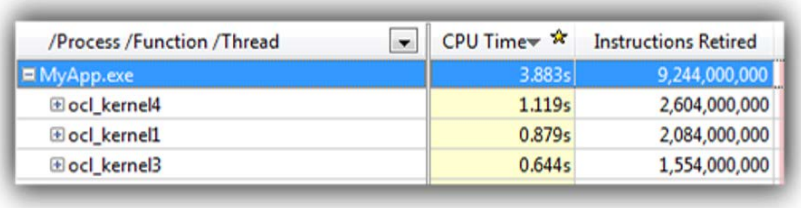
## 5.1.2 Viewing the OpenCL Kernel's Assembly Code

To view the OpenCL Kernel's Assembly Code, do the following:

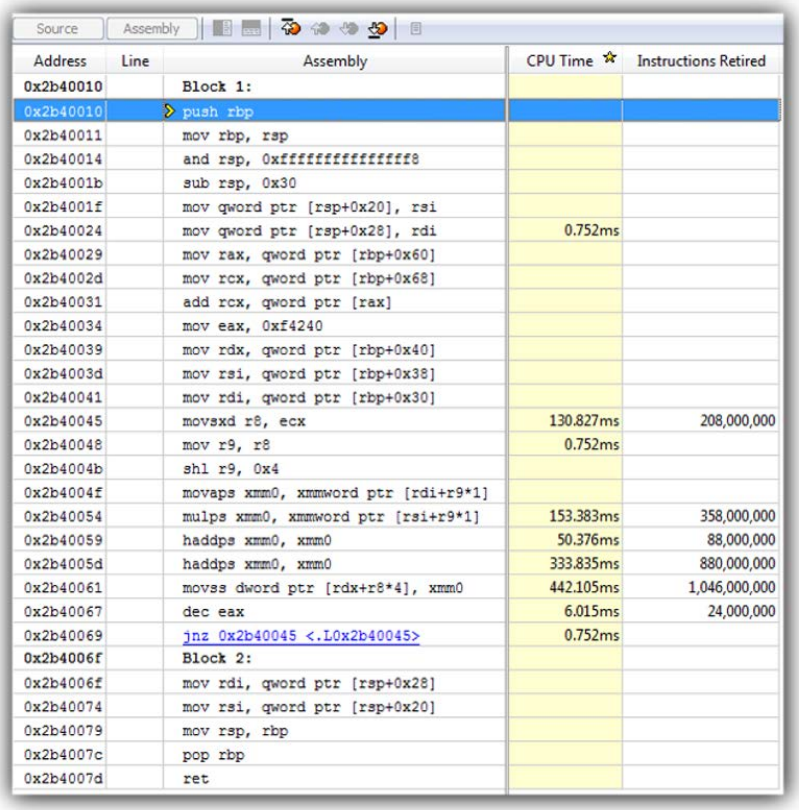
1. Wait until the sampling activity finishes.
2. Click on the **Hotspots Bottom-up** button at the navigation toolbar.
3. Select the **/ Function** option in the **Data Grouping** selection box and look for your application's OpenCL kernels in the **Functions** data grid:




If you are running several applications with identical OpenCL kernel names simultaneously, select the **/ Process / Function / Thread** option in the Data Grouping selection box and look for the application's process in the **Processes** data grid. The OpenCL kernel that belongs to the application is located under the application name in the application tree view, see the example below:



4. Double click the selected OpenCL kernel to see the assembly source code and its relevant sampling information:

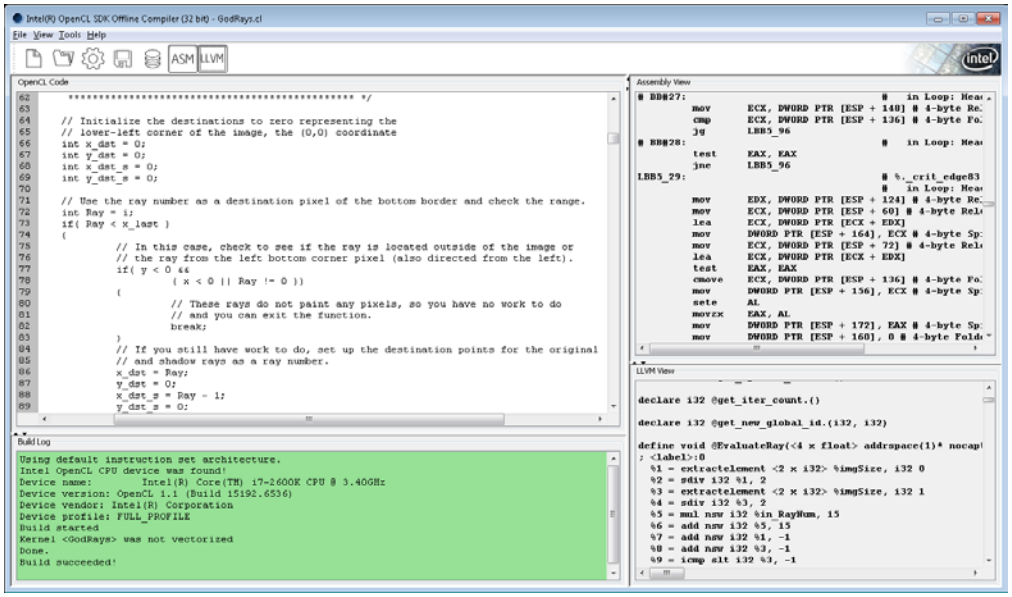



| Address | Line | Assembly | CPU Time | Instructions Retired |
|---|---|---|---|---|
| 0x2b40010 | | Block 1: | | |
| 0x2b40010 | | push rbp | | |
| 0x2b40011 | | mov rbp, rsp | | |
| 0x2b40014 | | and rsp, 0xfffffffffffffff8 | | |
| 0x2b4001b | | sub rsp, 0x30 | | |
| 0x2b4001f | | mov qword ptr [rsp+0x20], rsi | | |
| 0x2b40024 | | mov qword ptr [rsp+0x28], rdi | 0.752ms | |
| 0x2b40029 | | mov rax, qword ptr [rbp+0x60] | | |
| 0x2b4002d | | mov rcx, qword ptr [rbp+0x68] | | |
| 0x2b40031 | | add rcx, qword ptr [rax] | | |
| 0x2b40034 | | mov eax, 0xf4240 | | |
| 0x2b40039 | | mov rdx, qword ptr [rbp+0x40] | | |
| 0x2b4003d | | mov rsi, qword ptr [rbp+0x38] | | |
| 0x2b40041 | | mov rdi, qword ptr [rbp+0x30] | | |
| 0x2b40045 | | movsxd r8, ecx | 130.827ms | 208,000,000 |
| 0x2b40048 | | mov r9, r8 | 0.752ms | |
| 0x2b4004b | | shl r9, 0x4 | | |
| 0x2b4004f | | movaps xmm0, xmmword ptr [rdi+r9*1] | | |
| 0x2b40054 | | mulps xmm0, xmmword ptr [rsi+r9*1] | 153.383ms | 358,000,000 |
| 0x2b40059 | | haddps xmm0, xmm0 | 50.376ms | 88,000,000 |
| 0x2b4005d | | haddps xmm0, xmm0 | 333.835ms | 880,000,000 |
| 0x2b40061 | | movss dword ptr [rdx+r8*4], xmm0 | 442.105ms | 1,046,000,000 |
| 0x2b40067 | | dec eax | 6.015ms | 24,000,000 |
| 0x2b40069 | | jnz 0x2b40045 <.L0x2b40045> | 0.752ms | |
| 0x2b4006f | | Block 2: | | |
| 0x2b4006f | | mov rdi, qword ptr [rsp+0x28] | | |
| 0x2b40074 | | mov rsi, qword ptr [rsp+0x20] | | |
| 0x2b40079 | | mov rsp, rbp | | |
| 0x2b4007c | | pop rbp | | |
| 0x2b4007d | | ret | | |

## 5.2 Compiling Your Program with the Intel® OpenCL SDK Offline Compiler

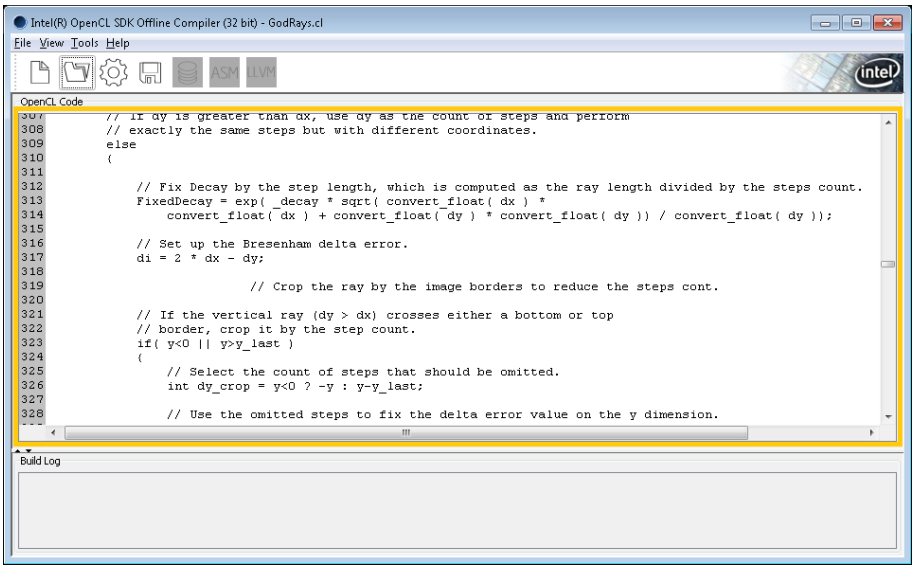
Intel® OpenCL SDK provides a unique tool which offers full offline OpenCL language compilation, including an OpenCL syntax checker, cross hardware platform compilation support, Low Level Virtual Machine (LLVM) viewer, Assembly language viewer and intermediate program binaries generator.

The tool supports Microsof* Windows* and Linux* operation systems. For detailed list of supported operation system see Technical Requirements chapter in the Intel® OpenCL SDK Release Notes document [see Related Information].



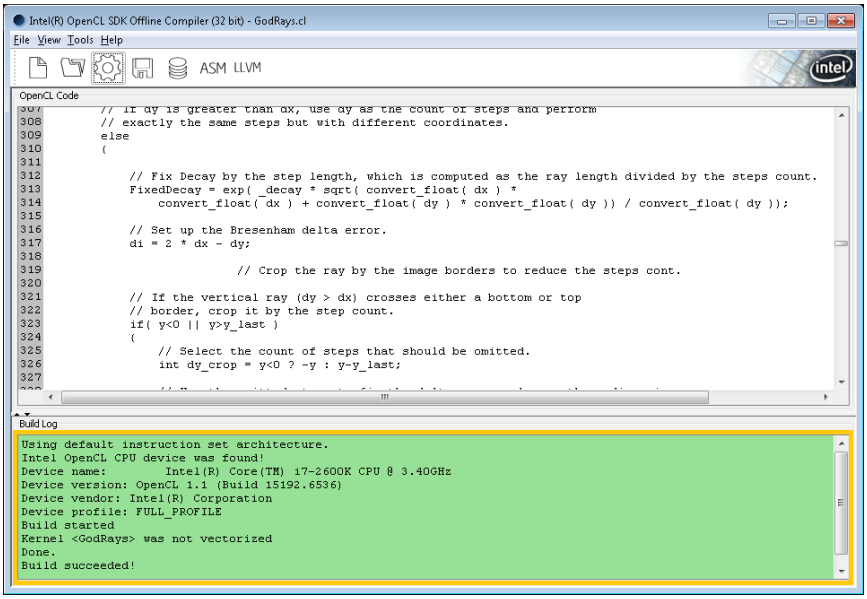

## 5.2.1 Building the OpenCL Kernels

1. Use the **OpenCL Code** text box to write your OpenCL code, or load the code from a file by clicking the **Load From File** button on the tool-bar, or by selecting **File** > **Load From File…**.


Intel(R) OpenCL SDK Offline Compiler (32 bit) - GodRays.cl
File View Tools Help
OpenCL Code
// If dy is greater than dx, use dy as the count of steps and perform
// exactly the same steps but with different coordinates.
else
{
// Fix Decay by the step length, which is computed as the ray length divided by the steps count.
FixedDecay = exp( _decay * sqrt( convert_float( dx ) *
convert_float( dx ) + convert_float( dy ) * convert_float( dy )) / convert_float( dy ));
// Set up the Bresenham delta error.
di = 2 * dx - dy;
// Crop the ray by the image borders to reduce the steps cont.
// If the vertical ray (dy > dx) crosses either a bottom or top
// border, crop it by the step count.
if( y<0 || y>y_last )
{
// Select the count of steps that should be omitted.
int dy_crop = y<0 ? -y : y-y_last;
// Use the omitted steps to fix the delta error value on the y dimension.
Build Log

2. Click the **Build** button on the commands tool-bar, or select **Tools** > **Build** from the main menu to build the OpenCL code.

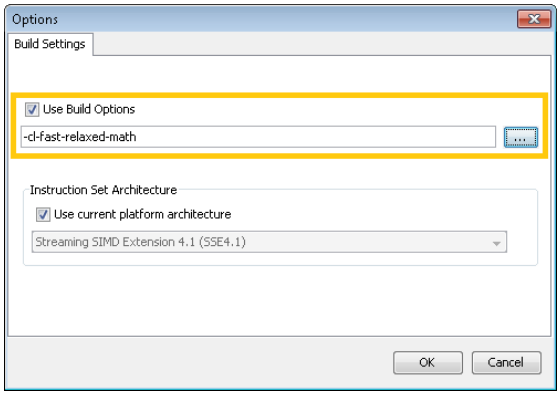
3. Look at the **Build Log** text box below to see information on the build status. If the build succeeds, the text box background color turns green, otherwise, it turns red.


Intel(R) OpenCL SDK Offline Compiler (32 bit) - GodRays.cl
File View Tools Help
ASM LLVM
OpenCL Code
// exactly the same steps but with different coordinates.
else
{
// Fix Decay by the step length, which is computed as the ray length divided by the steps count.
FixedDecay = exp( _decay * sqrt( convert_float( dx ) *
convert_float( dx ) + convert_float( dy ) * convert_float( dy )) / convert_float( dy ));
// Set up the Bresenham delta error.
di = 2 * dx - dy;
// Crop the ray by the image borders to reduce the steps cont.
// If the vertical ray (dy > dx) crosses either a bottom or top
// border, crop it by the step count.
if( y<0 || y>y_last )
{
// Select the count of steps that should be omitted.
int dy_crop = y<0 ? -y : y-y_last;
Build Log
Using default instruction set architecture.
Intel OpenCL CPU device was found!
Device name: Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
Device version: OpenCL 1.1 (Build 15192.6536)
Device vendor: Intel(R) Corporation
Device profile: FULL_PROFILE
Build started
Kernel <GodRays> was not vectorized
Done.
Build succeeded!

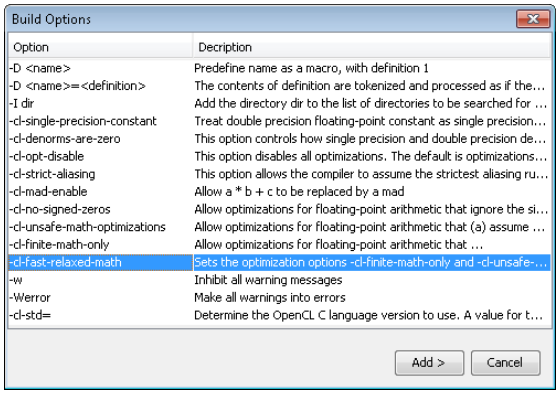
### 5.2.1.1 Configuring the Build Options

To configure the build options for the OpenCL code, do the following:

1. Select **Tools** > **Options...**
2. In the **Options** window select the **Use Build Options** check box and enter the build options in the **Build Options** text box:




Click on the **Browse** button next to the **Build Options** text box to see the full list of options supported by the OpenCL standard.


Build Options
Option
Decription
-D <name>
Predefine name as a macro, with definition 1
-D <name>=<definition>
The contents of definition are tokenized and processed as if the...
-I dir
Add the directory dir to the list of directories to be searched for ...
-cl-single-precision-constant
Treat double precision floating-point constant as single precision...
-cl-denorms-are-zero
This option controls how single precision and double precision de...
-cl-opt-disable
This option disables all optimizations. The default is optimizations...
-cl-strict-aliasing
This option allows the compiler to assume the strictest aliasing ru...
-cl-mad-enable
Allow a * b + c to be replaced by a mad
-cl-no-signed-zeros
Allow optimizations for floating-point arithmetic that ignore the si...
-cl-unsafe-math-optimizations
Allow optimizations for floating-point arithmetic that (a) assume ...
-cl-finite-math-only
Allow optimizations for floating-point arithmetic that ...
-cl-fast-relaxed-math
Sets the optimization options -cl-finite-math-only and -cl-unsafe-...
-w
Inhibit all warning messages
-Werror
Make all warnings into errors
-cl-std=
Determine the OpenCL C language version to use. A value for t...
Add >
Cancel

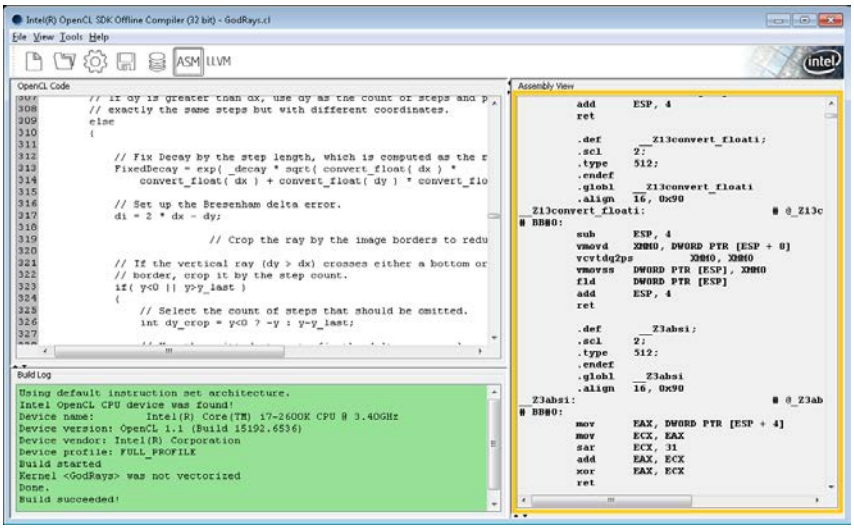
Select the rows with build options you want to add from the **Build Options** table and click **Add**. The options appear in the **Build Options** text box.

3. Click **OK** to return to the main form.

## 5.2.2 Viewing the Generated Assembly Code

The Intel® OpenCL SDK Offline Compiler tool enables you to see the generated assembly code of the OpenCL code.

1. After the build successfully completes, click the **View Assembly Code** ASM button in the toolbar or select **View** > **Show Assembly Code** from the main menu. The assembly code appears in the **Assembly Code** text box, to the left of the **OpenCL Code** text box.

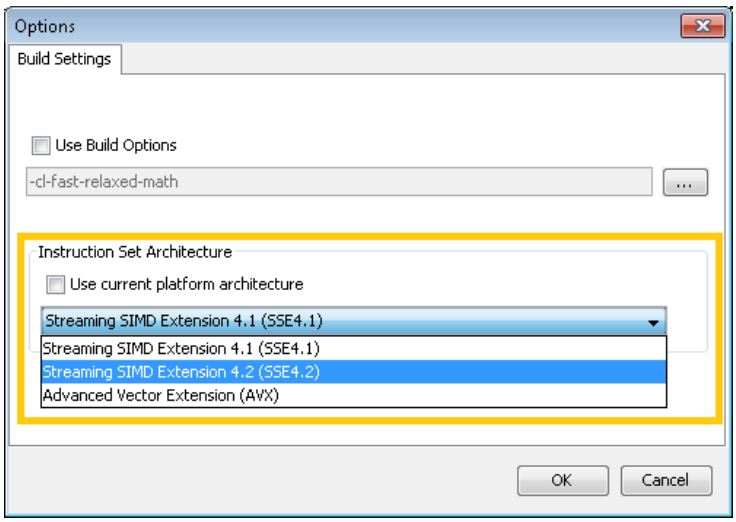



2. Click the **View Assembly Code** button again to hide the **Assembly Code** text box.

### 5.2.3 Choosing a Different Target Instruction Set Architecture

The Intel® OpenCL SDK Offline Compiler tool enables you to choose the target instruction set architecture when building an OpenCL code. It enables you to see the assembly code of different instruction set architectures and to generate program binaries for different hardware platforms.

1. Select **Tools** > **Options...**
2. In the **Options** window, under the **Instruction Set Architecture** group box uncheck the **Use current platform architecture** checkbox and select the appropriate ISA from the combo box below. The available items are:
    - Streaming SIMD Extension 4.1 (SSE4.1)
    - Streaming SIMD Extension 4.2 (SSE4.2)
    - Advanced Vector Extension (AVX)

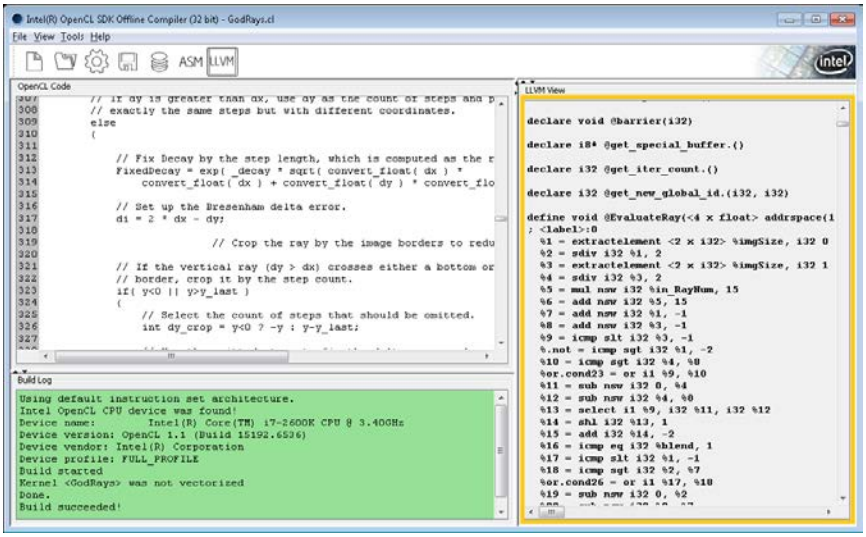



3. Click **OK** to return to the main window. The name of the target ISA appears on the main windows top bar as an indicator, next to the file name.

### 5.2.4 Viewing the Generated LLVM Code

To see the generated LLVM (Low Level Virtual Machine) IR from the OpenCL code, follow these steps:

1. When the build successfully completes, click the **LLVM** button on the application toolbar or select **View** > **Show LLVM Code**. The LLVM code appears in the **LLVM Code** text box, to the right of the **OpenCL Code** text box.




3. Click the **View LLVM Code** button again to hide the **LLVM Code** text box.

## 5.2.5 Generating Intermediate Program Binaries

The Intel® OpenCL SDK Offline Compiler tool enables you to generate program binaries of the OpenCL code. An application can use generated program binaries to create program from binaries later (`clCreateProgramWithBinary(…)`).

1. After the build successfully completes, click the **Create Program Binary** button or select **Tools** > **Create Program Binary**.
2. Choose the location and the name of the program binary in the **Save As** dialog box and click **OK** to save the file.

## 5.2.6 Saving Your Code to a Text File

The Intel® OpenCL SDK Offline Compiler tool enables you to save the generated Assembly/LLVM code, as well as the OpenCL code you entered.

Click the **Save As...** button and select the type of code which you want to save (Assembly/LLVM/OpenCL) or select **File** > **Save**.

### 5.2.7 Working with the Command Prompt

The Intel® OpenCL SDK Offline Compiler provides a command line utility. There are two versions of the utility, a 32-bit version (`ioc32.exe`) and a 64-bit version (`ioc64.exe`). The command-line tool is located in the following directory:

- `$(INTELOCLSDKROOT)\bin\x86` - for the 32-bit version of the tool
- `$(INTELOCLSDKROOT)\bin\x64` - for the 64-bit version of the tool.

Use the command `ioc32 -help` or (`ioc64 -help`) to view help information on all available switches in the command window.

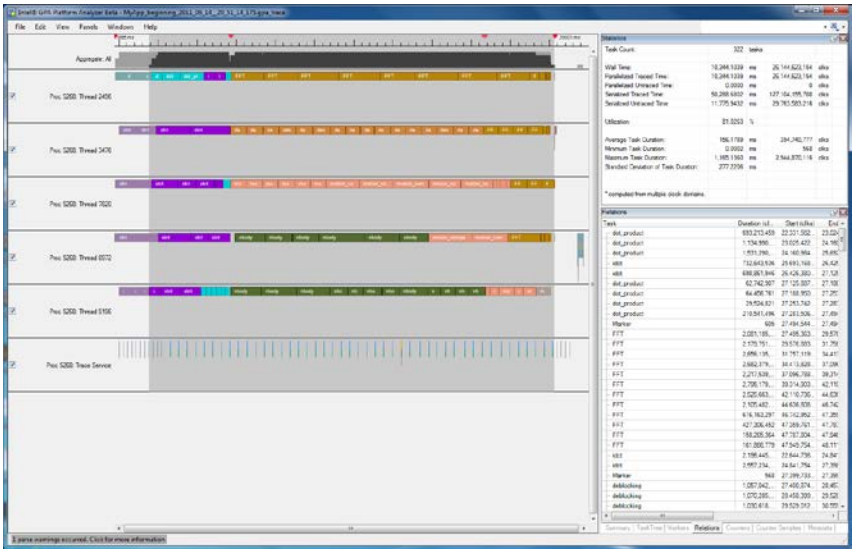
## 5.3 Tracing OpenCL Commands with the Intel® Graphics Performance Analyzers

The Intel® Graphics Performance Analyzers (Intel® GPA) Platform Analyzer provides a visual display of the execution profile of the various tasks in your code over a period of time. The tool collects real-time trace data during the application run, and provides a system-wide picture of the way your code executes on various CPU and GPU cores in your system. The tool infrastructure automatically aligns clocks across all cores in the entire system so that you can analyze CPU-based workloads together with GPU-based workloads within a unified time domain.

Intel® GPA is available for the Microsoft* Windows* operation systems only.

To download Intel® GPA, go to: http://software.intel.com/en-us/articles/intel-gpa/.

***NOTE:*** Intel® OpenCL SDK is designed to work with Intel® Graphics Performance Analyzer version 4.0 or higher.



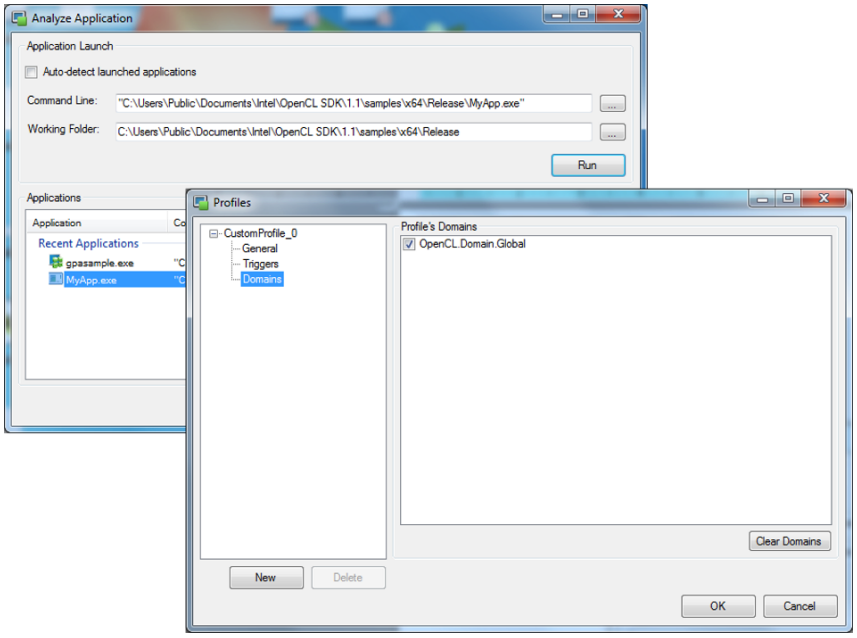
Intel® OpenCL SDK 1.5 provides built-in capturing support, with several profiling features such as:

- **Device view** which enables you to see the distribution of the application OpenCL commands (kernels and memory operations) across the system's software threads.
- **Context view** (available on Intel® GPA 4.2 only) which enables you to examine the flow of the OpenCL commands and their dependencies within the application's context command queues.
- **API tracing** which enables you to capture and measure the time of the application's OpenCL API calls.

You can use the generated trace file later to analyze the flow of execution (identify critical bottlenecks, and other information on the execution flow) and to improve its performance.

### 5.3.1 Generating a Trace File

After creating an empty platform analyzer trace file (see Intel® GPA Getting Started Guide for instructions), you should open **Profiles** window, click on **Domain** and, in the profiles domain section, select the desired domain.




To generate a trace file follow these steps:

1. Set a new environment variable named `CL_CONFIG_USE_GPA` to the value **True** (setting the environment variable back to **False** disables the tracing).
2. Run the Intel® GPA Monitor application.
3. Check the **Auto-detect launched application** check box in the **Application Launch** group box to automatically detect the OpenCL Application, or explicitly add the full path of the executable in the **Command Line** text box and click on the **Run** button to run the application.
4. Once the application name appears in the **Application** group box, right click on the application name within the **Running Applications** group and select **Create/Edit profile...** from the context menu.
5. In the new opened **Profiles** form, click the **New** button to create a new profile. Make sure that both **Enabled** and **Capture Application Startup** properties are set to the value **True**.
6. Run your OpenCL application.
7. When the application ends, the trace file is located under the output folder of Intel® GPA trace files, for example:

```
C:\Users\<user name>\Documents\GPA_4.0
```

where <user name> is the name of your local user account.

You can load the generated OpenCL trace data file into the Intel® GPA application. After loading the trace file, tasks and markers items display.

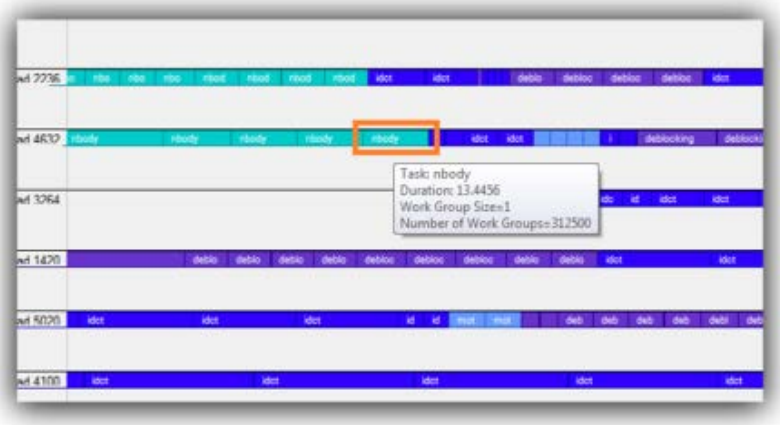
## 5.3.2 OpenCL Device View

The OpenCL Device View enables you to see the distribution of the application's OpenCL commands (kernels and memory operations) across the system's software threads.

### 5.3.2.1 Tasks

Two types of tasks are available on the timeline tracker:

- The OpenCL Kernels. Each task on the timeline tracker represents a cluster of workgroups which belongs to the same kernel. All tasks which belong to the same kernel are marked is the same color; each kernel has a different color.




- Memory operations. In addition to the OpenCL kernels, there are tasks on the timeline tracker which represent memory operations: Read, Write, Copy, Map and Un-map. All of these commands are red colored (▬).

### 5.3.2.2 Markers

For each command in the queue, the trace enables you to see a marker on the time ruler, indicating the state of the command. The following markers are supported:

- A marker indicating the time command was entered to the commands queued.
- A marker indicating the time command submitted to the device
- A marker indicating the time command started running.

- A marker indicating the time command completed.

The default view shows the queued and the completed markers only. To add other markers or remove existing ones, see Controlling the Markers Display.

## 5.3.3 Controlling the Markers Display

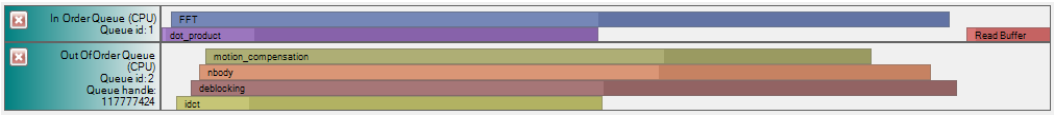
You can change the markers display in the time ruler by adding and modifying the following environment variables:

| Environment Variable | Default Value |
|---|---|
| CL_GPA_CONFIG_SHOW_QUEUED_MARKER | True |
| CL_GPA_CONFIG_SHOW_SUBMITTED_MARKER | False |
| CL_GPA_CONFIG_SHOW_RUNNING_MARKER | False |
| CL_GPA_CONFIG_SHOW_COMPLETED_MARKER | True |

## 5.3.4 OpenCL Context View

The OpenCL Context View enables you to examine the flow of the OpenCL commands and their dependencies within the application's context command queues.

You can view detailed timing information about your OpenCL workload by inspecting the time required to queue commands for the device, and the time the device took to execute items from the queue.




It enables you to know which commands took more time to execute.

Each tracker in the context view represents an OpenCL commands queue. These trackers can be easily identified by the turquoise ( ) color of their left headers. Two types of trackers exist:

- in-order queue (representing in-order OpenCL commands queue)
- Out Of Order queue (representing out-of-order OpenCL commands queue).

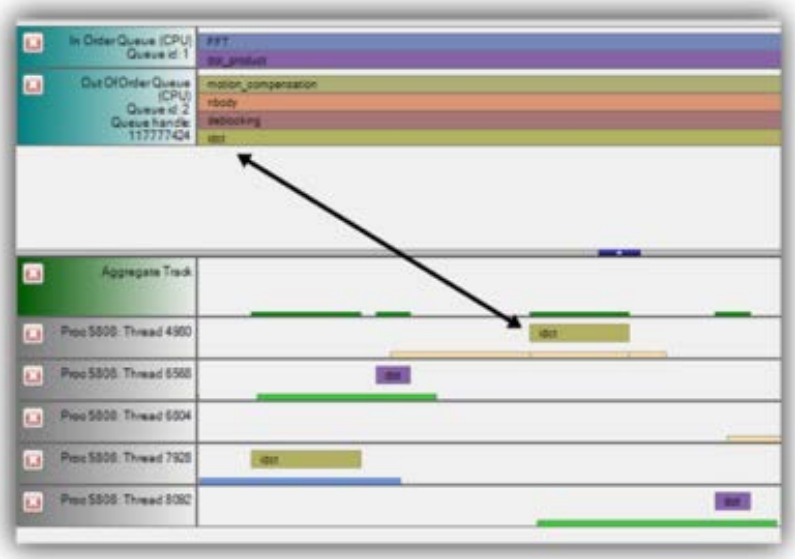
The tasks within these trackers represent OpenCL commands (OpenCL kernels or memory operations). All commands have different colors.

Eash task is logically divided into two parts:

- The wait time of the OpenCL command in the queue, before scheduling it for execution on the device.
- The actual execution time of the OpenCL command.

The two parts share the same color, although with different brightness levels.

The colors of the commands in the context view trackers are identical to the colors of the corresponding commands in the device view trackers.




## 5.3.5 OpenCL API Tracing

The OpenCL API Tracing feature enables you to view detailed timing information about your OpenCL workload by inspecting the time the host API took to execute. It enables you to know when certain parts of the Host API took more time to execute.

The OpenCL API Tracing is disabled by default. To enable it you need to control the following environment variable:

| Environment Variable | Default Value |
|---|---|
| CL_GPA_CONFIG_ENABLE_API_TRACING | True |

# 5.4 Using the Intel® OpenCL SDK Debugger

The Intel® OpenCL SDK Debugger is a Microsoft* Visual Studio* 2008 plug-in which enables you to debug into OpenCL kernels using the familiar graphical interface of the Microsoft* Visual Studio* 2008 debugger.

***NOTE:*** The Intel® OpenCL SDK Debugger provides a seamless debugging experience across host and OpenCL code, by supporting host code debugging and OpenCL kernel debugging in a single Microsoft* Visual Studio* debug session. The Intel® OpenCL SDK Debugger works with Microsoft* Visual Studio* 2008 only. Another version of the Microsoft* Visual Studio* is not supported. You must acquire Microsoft* Visual Studio* 2008 separately. For more information, see the Visual Studio* 2008 page at http://www.microsoft.com/visualstudio/en-us/products/2008-editions/.

## 5.4.1 Debugging Your OpenCL Kernel with Intel® OpenCL SDK Debugger

***NOTE:*** To work with the Intel® OpenCL SDK Debugger plug-in, the OpenCL kernel code must exist in a text file separate from the code of the host. Debugging OpenCL code which only appears in a string embedded in the host application is not supported.

To debug an OpenCL kernel, follow these steps:

1. Enable debuging mode in the Intel® OpenCL runtime for compiling the OpenCL code: add the `–g` flag to the **build options** string parameter in the `clBuildProgram` function.

2. Specify the full path of the file in the **build options** string parameter to the `clBuildProgram` function accordingly:

   ```
   -s <full path to the OpenCL source file>
   ```

   If there are spaces in the path you should enclose the entire path with double quotes (“”).

   For example:

   ```
   err = clBuildProgram(
             g_program,
             0,
             NULL,
   ```

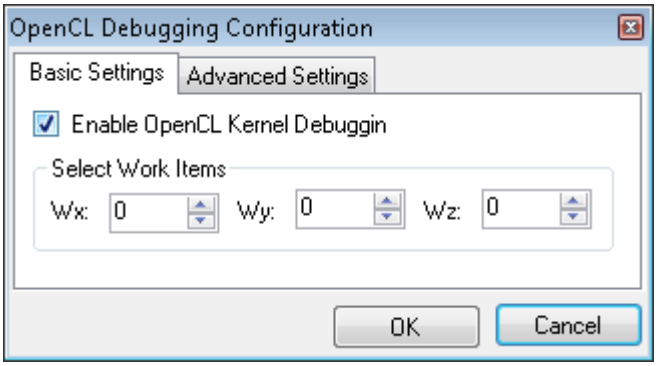
```
        "-g -s \"<path_to_opencl_source_file>\"",
        NULL,
        NULL);
```

According to the OpenCL standard, many work items execute the OpenCL kernels simultaneously. The Intel® OpenCL SDK Debugger requires setting in advance the global ID of the work item which you want to debug, before debugging session starts. The debugger stops on breakpoints in OpenCL code only when pre-set work item reaches them.

## 5.4.2 Configuring Intel® OpenCL SDK Debugger

To configure the Intel® OpenCL SDK Debugger, open the Debugging Configuration window:

1. Run Microsoft* Visual Studio* 2008.
2. Select **Tools** > **Intel OpenCL SDK Debugger**.



In the **Basic Settings** group box:

- Check the **Enable OpenCL Kernel Debugging** check box to switch Intel® OpenCL SDK Kernel Debugger on\off.
- Enter the appropriate values in the **Select Work Items** field to select work items.

  You can select only one work item. The values specify its 3D coordinates.

  If a NDRange running in less than 3D (i.e 1D or 2D), you must leave other dimensions at 0.

## 5.4.3 Troubleshooting the Intel® OpenCL SDK Kernel Debugger

***NOTE:*** The Intel® OpenCL SDK Debugger needs a local TCP/IP port to work correctly. On some occasions, you may encounter a problem for the debugger to use this port, due to a collision with another application or your firewall program.

***NOTE:*** If you receive "*Protocol error. If the problem continues, try changing Intel OpenCL kernel debugger port*" message, you may need to change the debugging port number and/or change your firewall settings.

To change the debugging port number, do the following:

1. Open **OpenCL Debugging Configuration** window
2. Switch to **Advanced Settings** group box.
3. Check the **Use Custom Debugging Port** check box.
4. In the **Debugging Port Number** field enter the port you need.

Intel® OpenCL SDK Kernel Debugger uses port 56203 by default.